Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 219 (2008) 67–82

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

A Rewriting Calculus for Multigraphs with Ports

Oana Andrei[1] and Hélène Kirchner[2]

INRIA & LORIA[3]

Abstract

In this paper, we define labeled multigraphs with ports, a graph model which specifies connection points for nodes and allows multiple edges and loops. The dynamic evolution of these structures is expressed with multigraph rewrite rules and a multigraph rewriting relation. Then we encode the multigraphs and multigraph rewriting using algebraic terms and term rewriting to provide an operational semantics of the multigraph rewriting relation. This term version can be embedded in the rewriting calculus, thus defining for labeled multigraph transformations a high-level pattern calculus, called ρ_{mg}-calculus.

Keywords: Multigraphs with ports, multigraph rewriting, term rewriting, rewriting calculus.

1 Introduction

Graphs are high-level constructs widely used for describing complex structures, like communication networks, neural networks, UML diagrams, microprocessor design, XML documents, biological systems. Graph transformation provides a rule-based modeling of their dynamic evolution. Different approaches have been proposed to formalize graph transformation and to define graph rewriting, summarized for instance in [18].

We have explored graph models for simulating chemical reactors [8,1] and protein interactions [3]. In this context we found the need for graph structures where the nodes have points, called *ports*, for attaching the edges, thus providing an explicit partitioning of nodes connectivity. We have identified a quite general class of directed graphs allowing multiple edges and loops, where node information is represented as node labels, and an edge label is the ordered pair of source port and target port; we call such graphs *labeled multigraphs with ports*.

[1] Email: Oana.Andrei@loria.fr

[2] Email: Helene.Kirchner@loria.fr

[3] UMR 7503 CNRS-INPL-INRIA-Nancy2-UHP. Campus Scientifique, BP 239, Nancy, France

doi:10.1016/j.entcs.2008.10.035

The concept of port for graphs is not a novelty. It can be seen as a refinement of the connectivity information for nodes. The Graph Markup Language GraphML [9], an XML format for graph structures, backed to the graph drawing community, uses ports in nodes for partitioning incidences. An immediate application of multigraphs with ports is for modeling protein-protein interactions concerned with the connectivity inside molecular complexes (see [13] for a process algebra approach, and [7] for an approach based on graph rewriting). Proteins are abstracted as boxes with interaction sites on the surface having particular states. Hence, adding a refinement on the ports and calling them sites with at most one edge attached to each port, the multigraph rewriting and its correspondent rewriting calculus become suitable for modeling the interactions of molecular complexes. We called this variant of multigraphs with ports used for modeling molecular complexes *molecular graphs* [3]. In Fig. 1 we illustrate in the middle a reaction pattern that applied on the left molecular graph creates an edge (bond) as we can see in the molecular graph on the right. This example is extracted from a larger example developed in [3] which models the beginning of the epidermal growth factor receptor signaling cascade. The protagonists of the example are four signal proteins denoted by **S** with **S.S** their dimerized form, two receptor proteins **R**, and one adapter protein **A**. Sites are represented differently according to their state: filled circles for bound sites, and empty circles for free ones.

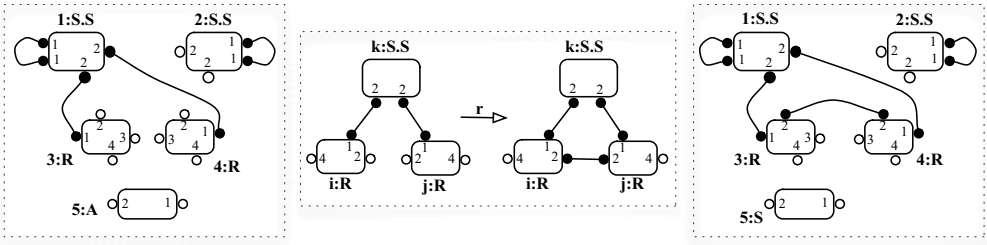


Fig. 1. Two molecular graphs related by a complexation reaction

Membranes can also form complexes, called tissues, due to the binding proteins on their surfaces. While on the one side we can model interactions between biochemical entities like proteins, proteins and lipids, or membranes, on the other side we are able to model as well chemical reactions (like the ones in [1]) using the multigraph rewriting: atoms represent the nodes, the covalence of an atom gives the number of identical ports, and chemical bonds between atoms are multiple edges.

The paper is divided in three parts: after giving basic definitions in Sect. 2, we first define in Sect. 3 the labeled multigraphs with ports, multigraph rewrite rules, and the multigraph rewriting relation; second, in Sect. 4, we encode the multigraphs and multigraph rewriting using algebraic terms and term rewriting to provide an operational semantics for the multigraph rewriting relation; and third, in Sect. 5, we embed the term approach on multigraph rewriting in the rewriting calculus obtaining for free a rewriting calculus for labeled multigraphs, called ρ_{mg}-calculus. Therefore we provide for the multigraph rewriting a high-level calculus extending

algebraic rewriting allowing us to benefit from the properties of the ρ-calculus, especially the possibility of using rewriting strategies and rule conditions to control rule application. The operational correspondence result stated in Sect. 4 allows us to visually express quite complex multigraph transformations and perform them using term rewriting. In Sect. 6 we give some implementation hints for multigraph rewriting and sketch some extensions and applications for multigraphs with ports.

The proofs of the results stated in this paper are available in [2].

2 Background

In this section we briefly review some basic definitions of graph theory and graph transformation [12,18] used in this paper. We adopt the classical definitions for order-sorted algebra and term rewriting from [14] and [4,15] respectively.

Labeled Graphs. A *label alphabet* $\mathcal{L} = (\mathcal{L}_V, \mathcal{L}_E)$ is a pair of sets of node labels and edge labels. A (finite) *graph* over $\mathcal{L}$ is a triple $G = (V, E, s, t, l)$ where V is a set $\{v_1, \ldots, v_k\}$ of elements called *nodes* (or *vertices*), E is a set $\{e_1, \ldots, e_m\}$ of elements of the Cartesian product $V \times V$ called *edges*, $s, t : E \to V$ are the source ant target functions respectively, and $l = (l_V, l_E)$ is the *labeling function* for nodes $(l_V : V \to \mathcal{L}_V)$ and edges $(l_E : E \to \mathcal{L}_E)$. If G is a graph, we usually denote by V_G its node set and by E_G its edge set. An edge of the form (v, v) is called a *loop*. For an edge (u, v), u and v are called *end nodes* with u the *source* and v the *target*; moreover we say that u and v are *adjacent* or *neighbouring* nodes, with v neighbour of u. An edge is *incident* to a node if the node is one of its end nodes. An edge is *multiple* if there is another edge with the same source and target; otherwise it is *simple*. A *multigraph* is a graph allowing multiple edges and loops. An *adjacency list* for a node is given by a list of pairs consisting of a neighbour and the corresponding edge label. If a node has no neighbour then its adjacency list is empty. A *subgraph* of a graph G is a graph whose node and edge sets are subsets of those of G. A *graph morphism* assigns the nodes and edges of a given graph to the nodes and edges of another graph while preserving adjacency. In the case of labeled graphs, the node and edge labeling is also preserved.

Graph Transformation. A *graph transformation rule* $L \rightsquigarrow R$ consists of two graphs L and R called the left- and right-hand side respectively, and a correspondence between elements of the left-hand side and elements of the right-hand side. This correspondence is provided by some unique identifiers associated to nodes.

As presented in [18], the application of a graph transformation rule $L \rightsquigarrow R$ to a graph G, called *host graph*, produces a new graph G' according to the following steps:

(i) Find a matching morphism m for L in G (hence $m(L)$ is a subgraph of G).

(ii) Remove the subgraph $m(L)$ from G resulting in the context graph G^-.

(iii) Add $m(R)$ to the context graph G^-.

(iv) Reconnect $m(R)$ and G^-.

The differences between various approaches for graph replacement arise mainly

in the last step, depending on the mechanism chosen for establishing connections between new and old nodes. Two particular problems are handled at this stage ([12,18]). The first one refers to whether or not noninjective matching is allowed. For example, if two different nodes L are matched to one node in the host graph, and one of the two nodes is deleted and the other preserved, will the node in the host graph be deleted or preserved? The second problem concerns the dangling edges in the host graph which are unmatched edges with one endpoint deleted by the transformation rule. These two problematic situations are referred to as the identification and the dangling problem respectively. We will see later how we handle these points in our framework.

3 Labeled Multigraphs with Ports

We refine the definition of multigraphs by typing nodes with names and by adding explicit connection points, called *ports*, to nodes; then edges attach, more specifically, to ports of nodes. Let $\mathscr{P}$ be a finite set of ports and $\mathscr{N}$ a finite set of node names.

Definition 3.1 A *labeled multigraph with ports* over $\mathscr{N}, \mathscr{P}$ takes the form $G = (V, E, \iota, s, t, l)$ where:

- V is a finite set of nodes (also referred to as node identifiers);
- $\iota = (\iota_1, \iota_2) : V \to \mathscr{N} \times \mathcal{P}(\mathscr{P})$ assigns a name and a port set to each node, with $\iota_1(v) = n$ and $\iota_2(v) = P$ for $\iota(v) = (n, P)$;
- $E \subseteq \{(v^\frown p, u^\frown r) \in (V \times P)^2 \mid p \in \iota_2(v), r \in \iota_2(u)\}$ a finite multiset of edges;
- $s, t : E \to V \times P$ the usual source and target functions;
- $l = (l_V, l_E)$ is the labeling function associating to each node $v \in V$ the triple consisting of the identifier, the name, and the port set, $l_V(v) = \langle v : \iota_1(v) \parallel \iota_2(v) \rangle$, and to each edge $(v^\frown p, u^\frown r) \in E$ the couple formed by the source port and the target port, $l_E((v^\frown p, u^\frown r)) = (p, r)$.

Hereinafter we say (labeled) multigraph instead of labeled multigraph with ports if there is no risk of confusion. Let $\mathscr{P} = \{a, b, c, \ldots\}$ and $\mathscr{N} = \{A, B, C, \ldots\}$ the sets of constants denoting ports and names respectively. We consider variables ports and names as well, denoted by $\mathcal{X}_\mathscr{P} = \{x, y, z, \ldots\}$ and $\mathcal{X}_\mathscr{N} = \{X, Y, Z, \ldots\}$ respectively. We represent the node identifiers by non-empty sequences of integers. We denote by $\mathcal{V}ar(G)$ the set of variables occurring in G. In Fig. 2 we illustrate two views of a multigraphs with ports: on the left, we use the classical drawing of a labeled multigraph, while on the right, we emphasize the ports. We will use the latter more suggestive representation for multigraphs by representing a node as a box with the identifier and the name placed outside the box and a port as a small point on the surface of the box.

Graph transformation rules are instantiated in this context.

Definition 3.2 (Multigraph rewrite rule) A *multigraph rewrite rule* is an ordered pair of multigraphs over $\mathscr{N} \cup \mathcal{X}_\mathscr{N}, \mathscr{P} \cup \mathcal{X}_\mathscr{P}$ denoted by $L \rightsquigarrow R$, where all node

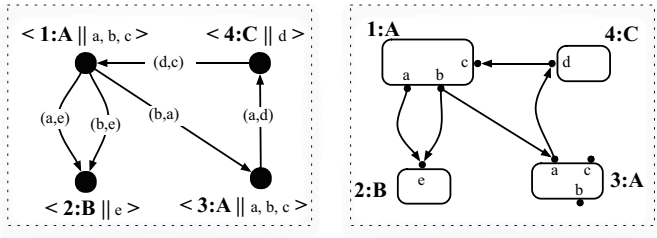


Fig. 2. Two views of a labeled multigraph with ports

identifiers in L are variables, and such that $\mathcal{V}ar(L) \supseteq \mathcal{V}ar(R)$. The multigraphs L and R are called the left- and right-hand side of the rule respectively.

Definition 3.3 (Node-substitution) The correspondence between nodes of the left- and right-hand side of a multigraph rewrite rule $L \rightsquigarrow R$ is a mapping $\xi : V_L \to \mathcal{P}(V_R)$ that we call *node-substitution*. It associates to each node v in L a (possibly empty) set of nodes in R. In practice, it associates to each node v in L the set of all nodes in R whose identifiers contain v; if v is deleted then it does not occur in the node identifiers in R and $\xi(v) = \emptyset$.

Given the definition above, we assume that for each multigraph rewrite rule we can automatically extract from the node identifiers the node-substitution.

Example 3.4 We illustrate in Fig. 3 four multigraph transformation rules: (a) splitting a node with $\xi(i) = \{i.1, i.2\}$, $\xi(j) = \{j\}$; (b) deleting a node with $\xi(i) = \emptyset$, $\xi(j) = \{j\}$; (c) deleting two edges with $\xi(v) = \{v\}$ for each node v; (d) merging two nodes $\xi(i) = \xi(j) = \{i.j\}$.

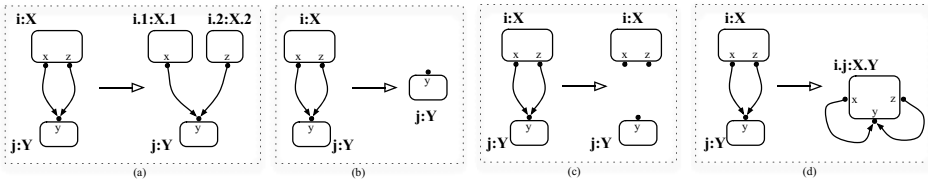


Fig. 3. Multigraph rewrite rules

We call a *partial node* of a node v with $\iota(v) = (n, P)$ a node with the same identifier, the same name, and a non-empty subset of P as the port set.

Let $L \rightsquigarrow R$ be a multigraph rewrite rule applied to the multigraph G with ξ the associated node-substitution. Then a matching morphism m for L in G assigns the nodes of L to partial nodes of G while preserving adjacency. Each matched node $v \in V_G$ with some unmatched ports can be partitioned into a matched partial node containing the matched ports (hence it occurs in $m(L)$), and an *unmatched partial node* containing the unmatched ports. A constant name can be mapped by a multigraph morphism only to itself.

The delicate point of applying $L \rightsquigarrow R$ to G is to properly define the replacement of $m(L)$ by $m(R)$ in G and the way $m(R)$ is reconnected with G. Let us first

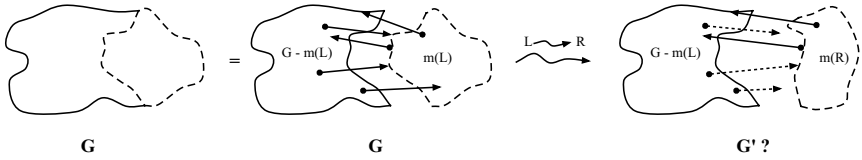


Fig. 4. A graph replacement sketch

illustrate graphically in Fig. 4 the replacement procedure: the first graph is G where the area with the dashed border represents $m(L)$; the second graph is also G but where we emphasize the edges (that we call *bridges*) connecting unmatched nodes to matched nodes; and the third graph represents the result of replacing the subgraph $m(L)$ by $m(R)$ and we see that some bridges are pending if their targets formerly in $m(L)$ no longer occur in $m(R)$. The step of reconnecting identifies the old end node of the bridges from $m(L)$ with their correspondent nodes from $m(R)$ given by $\xi(m)$. The same operation must be performed as well on the unmatched edges and unmatched partial nodes. Then, the result of the multigraph rewriting of G consists of putting together the context multigraph G^-, $m(R)$, and the updated unmatched edges and partial nodes, and bridges using $\xi(m)$. The unmatched partial nodes have to be updated first, since they can be end points for unmatched edges or bridges.

The *context multigraph* $G^- = G \setminus m(L)$ is given by the set of nodes $V_{G^-} = V_G \setminus V_{m(L)}$, and the set of edges $E_{G^-} = \{(u^\frown p, v^\frown r) \in E_G \mid u, v \in V_{G^-}\}$.

We denote by $\mathcal{U}_n$ the set of *unmatched partial nodes*, by $\mathcal{U}_e$ the set of *unmatched edges* (i.e., the not matched edges whose both endpoints are matched), and by $\mathcal{B}$ the set of *bridges* (i.e., the edges in G not matched by an edge of L, with one end a matched node and the other end not matched).

If $v \in \mathcal{U}_n$ and $\xi(v) = \{v_1, \ldots, v_k\}$ then, for each k-partition $(P_1, \ldots, P_k)$ of the port set $\iota_2(v)$, we have $m(\xi)(v) = \{m(v_1), \ldots, m(v_k)\}$ with $\iota_2(v_i) = P_i$. The application of $m(\xi)$ on a set of edges E is defined component-wise, and $m(\xi)((u^\frown p, v^\frown r)) = \{(u_k^\frown p, v_l^\frown r) \mid u_k \in m(\xi)(u) \text{ s.t. } p \in \iota_2(u_k) \text{ and } v_l \in m(\xi)(v) \text{ s.t. } r \in \iota_2(v_l)\}$.

By analogy with term rewriting, we can write $G = G^-[m(L)]_{\mathcal{U}_n,\mathcal{U}_e,\mathcal{B}}$, as a decomposition of G into the context graph G^-, the matched subgraph $m(L)$, the unmatched partial nodes $\mathcal{U}_n$, the unmatched edges $\mathcal{U}_e$, and the bridges $\mathcal{B}$. Then, once $m(R)$ is computed, it is replaced in the context multigraph G^- and the bridges are reconnected, thanks to $m(\xi)(\mathcal{U}_n)$, $m(\xi)(\mathcal{U}_e)$, and $m(\xi)(\mathcal{B})$, to get a resulting multigraph $G^-[m(R)]_{m(\xi)(\mathcal{U}_n),m(\xi)(\mathcal{U}_e),m(\xi)(\mathcal{B})}$.

We are now able to formulate the definition of multigraph rewriting.

Definition 3.5 (Multigraph rewriting relation) Given a multigraph rewrite system $\mathcal{R}$, a multigraph G rewrites to a multigraph G', denoted by $G \rightsquigarrow_{\mathcal{R}} G'$, if there exists:

- a multigraph rewrite rule $L \rightsquigarrow R$ in $\mathcal{R}$,
- a multigraph morphism m such that $m(L)$ is a subgraph of G,
- a set of unmatched partial nodes $\mathcal{U}_n$, a set of unmatched edges $\mathcal{U}_e$, and a set of

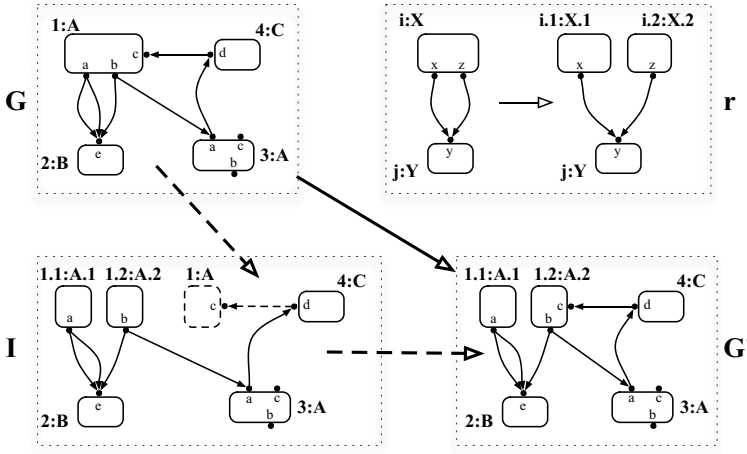


Fig. 5. An application of the multigraph rewrite rule r on G resulting in the multigraph G' with an intermediate multigraph I

bridges $\mathcal{B}$,

such that $G = G^-[m(L)]_{\mathcal{U}_n,\mathcal{U}_e,\mathcal{B}}$ and $G' = G^-[m(R)]_{m(\xi)(\mathcal{U}_n),m(\xi)(\mathcal{U}_e),m(\xi)(\mathcal{B})}$.

With respect to the discussion at the end of Section 2 concerning the identification and the dangling problems, the particularities of labeled multigraphs with ports transformations are first, that we consider only injective matching morphism, second, that when deleting a node, all its incident edges are deleted as well.

Example 3.6 We illustrate in Fig. 5 a multigraph resulting from rewriting G (also given in Fig. 2) using the rule r (also given in Fig. 3 (a)). The resulting multigraph G' is obtained by splitting the node 1, choosing to place the unmatched port c in 1.2, and then redirecting the two bridges $(4^\frown d, 1^\frown c)$ and $(1^\frown b, 3^\frown a)$ to 1.2. In the intermediate step we emphasize the incidence of the node 4 to the unmatched partial node 1 with the the port set $\{c\}$. The node-substitution may identify this partial node to either of the two resulting nodes 1.1 and 1.2. Therefore two solutions are possible: the one illustrated in Fig. 5, and the other one where c is placed in the node 1.1.

In order to define multigraph rewriting in a more operational way, we choose to go in the world of algebraic terms and term rewrite rules and to benefit from a classical ACU-matching algorithm for the application of rewrite rules, where ACU stands for associative-commutative with neutral element.

4 Term Rewriting Semantics for Multigraph Rewriting

In this section we define an encoding function $\mathcal{E}$ for multigraphs, multigraph rewrite rules, and multigraph rewrite relation as particular terms, term rewrite rules, and term rewriting relation respectively.

Concerning the problem of dangling edges, instead of mapping a node from the

$$\bullet : \longrightarrow Id \qquad \bullet : \longrightarrow Port \qquad \bullet : \longrightarrow Node \qquad \epsilon_X : \longrightarrow XSet$$
$$\_,\_ : XSet\ XSet \longrightarrow XSet\ [ACU(\epsilon_X)]$$
$$\langle \_ : \_ \parallel \_ \rangle : Id\ Name\ PortSet \longrightarrow Node$$
$$(\_,\_) : Port\ Port \longrightarrow Edge$$
$$\_ \frown \_ : Id\ EdgeSet \longrightarrow Neighbour$$
$$\_ \mathrel{\hat{=}} \_ : Id\ NeighbourSet \longrightarrow AdjacencyEq$$
$$\_(\!|\_|\!) : NodeSet\ AdjacencyEqSet \longrightarrow MGraph$$

Fig. 6. The operation set $\mathcal{F}$

left-hand side to an empty set of nodes from the right-hand side, we introduce a special node $\bullet$ called the *black hole*. Consequently, we replace a deleted node in an intermediate step by a black hole whose behaviour consists in deleting itself along with the incident edges. In a similar way we use the black hole to replace in an intermediary step a deleted port as well. Hence, the dangling edges are deleted using some particular intermediate and transparent operation as we will see later in this section.

We define an order-sorted signature $\Sigma = (\mathcal{S}, <, \mathcal{F})$ for encoding multigraphs, where, in order to eliminate redundancies, we represent a multigraph as a pair made of the set of node labels and the set of adjacency lists (for each node we list its neighbours with the corresponding edges as pairs of ports):

the sort set $\mathcal{S}$ consists of sorts for each component or set of components: *Id*, *Name*, *Port*, *Node*, *Edge*, *Neighbour*, *AdjacencyEq*, *PortSet*, *NodeSet*, *EdgeSet*, *NeighbourSet*, *AdjacencyEqSet*, *MGraph*.

the subsort relation is defined by $X < XSet$ for $X \in \{Port, Node, Edge, Neighbour, AdjacencyEq, MGraph\}$, i.e. each term of sort X can be seen as a set with a single element.

the operation set $\mathcal{F}$ allowing to describe the graph structure, is given in Fig. 6 where X takes sort values from the set $\{Node, Edge, Neighbour, AdjacencyEq\}$. The associative-commutative operator $\_,\_$ (union) is overloaded on each of the set sorts, and ϵ_X denotes the identity element (the empty set) for the operation $\_,\_$. We use ϵ instead of ϵ_X whenever the sort X can be easily deduced from the context. The constant operator $\bullet$ is overloaded as well, it can be an *Id*-, a *Port*- or a *Node*-sorted term.

Let $\mathcal{X}$ be an $(\mathcal{S}, <)$-sorted family of variables.

Definition 4.1 (Encoding multigraphs as terms) We encode a labeled multigraph $G = (V, E)$ as an algebraic term $\mathcal{E}(G) = T_1(\!|T_2|\!)$ of sort *MGraph* where:

- $T_1 \in \mathcal{T}_{\Sigma, NodeSet}(\mathcal{X})$ is the set of all node labels in G, and
- $T_2 \in \mathcal{T}_{\Sigma, AdjacencyEqSet}(\mathcal{X})$ is the set of adjacency equations providing the neighbours for each node in V (if any) and the pairs of ports corresponding to the incident edges.

Additionally, algebraic terms encoding multigraphs must satisfy some structural properties in order to be considered well-formed.

Definition 4.2 (Well-formed terms) A term $t \in \mathcal{T}_{\Sigma,MGraph}(\mathcal{X})$ is *well-formed* if (i) each node identifier occurs at most once: in the node set, in the adjacency equation set as left-hand side of an adjacency equation, in the neighbour set of a node identifier, (ii) each node identifier or port occurring in the adjacency equation set must also occur in the node set.

We also impose a canonical form (a representative of each equivalence class modulo ACU) for the terms encoding multigraphs, in order to eliminate useless information as follows:

Definition 4.3 (Canonical form) A term $t \in \mathcal{T}_{\Sigma,MGraph}(\mathcal{X})$ is in *canonical form* if:

- right-hand sides of adjacency equations are non-empty sets of neighbours;
- only non-empty set of edges occur in neighbour terms.

Example 4.4 The multigraph G illustrated in Fig. 2 is encoded as the following term: $\mathcal{E}(G) = (\langle 1 : A \parallel a, b, c\rangle, \langle 2 : B \parallel e\rangle, \langle 3 : A \parallel a, b, c\rangle, \langle 4 : C \parallel d\rangle)(\!| \ 1 \doteq (2^\frown(a, e), (b, e)), (3^\frown(b, a))), \ (3 \doteq 4^\frown(a, d)), (4 \doteq 1^\frown(d, c)) \ |\!)$.

For all rewrite rules over $\mathcal{T}_{\Sigma,MGraph}(\mathcal{X})$, according to Definition 3.2, we impose node identifiers occurring in the left-hand side to be variables. We say that a rewrite rule over $\mathcal{T}_{\Sigma,MGraph}(\mathcal{X})$ is well-formed (in canonical form) if both t_1 and t_2 are well-formed (in canonical form respectively). We call *mg-rewrite rule* a well-formed rewrite rule in canonical form.

Definition 4.5 (Encoding multigraph rewrite rules as term rewrite rules) Given a labeled multigraph rewrite rule $L \rightsquigarrow R$, we encode it as a term rewrite rule $\mathcal{E}(L \rightsquigarrow R) = \mathcal{E}(L) \twoheadrightarrow \mathcal{E}(R)$.

The encoding of a multigraph rewrite rule is an mg-rewrite rule since, by definition, the term encoding a multigraph is well-formed and in canonical form.

The *node-substitution* from nodes of the left-hand side to nodes of the right-hand side of an mg-rewrite rule can be extracted automatically by means of an analysis on the identifier occurrences. We call this procedure GetMap; it produces a set of elementary mappings (or elementary node-substitutions) from $\mathcal{T}_{\Sigma,Node}(\mathcal{X})$ to $\mathcal{T}_{\Sigma,NodeSet}(\mathcal{X})$ for each node occurring in the left-hand side of the rule. Identity mappings are usually omitted. Note that the node-substitution for a multigraph rewrite rule is encoded as in the following example.

Example 4.6 The encoding of the multigraph rewrite rule (a) given in Fig. 3 is:

$$(\langle i : X \parallel x, z\rangle, \langle j : Y \parallel y\rangle)(\!|i \doteq j^\frown(x, y), (z, y)|\!) \twoheadrightarrow$$

$$(\langle i.1 : X.1 \parallel x\rangle, \langle i.2 : X.2 \parallel z\rangle, \langle j : Y \parallel y\rangle)(\!|(i.1 \doteq j^\frown(x, y)), (i.2 \doteq j^\frown(z, y))|\!)$$

with the node-substitution $\xi = \{\langle i : X \parallel x, z\rangle \mapsto (\langle i.1 : X.1 \parallel x\rangle, \langle i.2 : X.2 \parallel z\rangle)),$ $\langle j : Y \parallel y\rangle \mapsto \langle j : Y \parallel y\rangle\}$.

After encoding multigraphs and their transformation rules, we now translate multigraph rewriting into term rewriting.

In a first step we customize the rewrite rules on $\mathcal{T}_{MGraph}(\mathcal{X})$ before applying them. In order to model multigraph rewriting using algebraic terms, we need to handle the context of the multigraph in which the replacement is performed. This is done by a systematic enrichment of rewrite rules with extension variables that help storing the context and applying rewrite steps in subterms. This is a usual method employed when performing rewriting modulo associativity and commutativity [16]. We usually denote by W an extension variable and by $\bar{t}$ the extension of term t. For each rewrite rule $t_1 \twoheadrightarrow t_2$, extension variables are appended to set-sorted terms to produce the extended rule $(\overline{t_1 \twoheadrightarrow t_2})$. An extension variable is added to each set-sorted subterm in the left-hand side (and accordingly in the right-hand side). This technical construction is formalised in [2]. We just give here an example of rule extension.

Example 4.7 The extension of the rule given in Example 4.6 is:
$(\langle i : X \parallel x, z, W_1^p, W_2^p\rangle, \langle j : Y \parallel y, W_3^p\rangle, W_4^n)(\!|(i \doteq (j^\frown((x,y),(z,y),W_5^e)), W_6^h), W_7^a|\!) \twoheadrightarrow$
$(\langle i.1 : X.1 \parallel x, W_1^p\rangle, \langle i.2 : X.2 \parallel z, W_2^p\rangle, \langle j : Y \parallel y, W_3^p\rangle, W_4^n)(\!|(i.1 \doteq j^\frown(x,y)),$
$(i.2 \doteq j^\frown(z,y)), (i \doteq (j^\frown W_5^e), W_6^h), W_7^a|\!)$
with the node-substitution $\xi = (\langle i : X \parallel x, z, W_1^p, W_2^p\rangle) \mapsto \langle i.1 : X.1 \parallel x, W_1^p\rangle, \langle i.2 : X.2 \parallel z, W_2^p\rangle), (\langle j : Y \parallel y, W_3^p\rangle) \mapsto \{\langle j : Y \parallel y, W_3^p\rangle)$ where the extension variables W_i have appropriate set sorts. The exponents used for the extension variables indicate their set sort: p for *PortSet*, n for *NodeSet*, e for *EdgeSet*, h for *NeighbourSet*, a for *AdjacencyEqSet*.

Instantiation of a node-substitution

Let σ be a substitution and ξ a node-substitution. We denote by ξ^σ the instantiation by σ of variables occurring in ξ computed component-wise: $\{t \mapsto t_1, \ldots, t_k\}^\sigma = \{\sigma(t) \mapsto \sigma(t_1), \ldots, \sigma(t_k)\}$.

Node-substitution application

The application of a node-substitution ξ on a term consists in applying sequentially each elementary node-substitution of ξ on the term; this is illustrated in Fig. 7. An elementary node-substitution is propagated inside an *MGraph*-sorted term using (Propagate), inside the set of adjacency equations of neighbours using (Distribute), and then applied on each of them. The application of a node-substitution on an adjacency equation using (ApplySrc) (or a neighbour using (ApplyTar)) transforms it in n adjacency equations (resp. neighbours), one for each corresponding node in the right-hand side of the mapping, and propagates the node-substitution application on the set of neighbours. We illustrate this operation in Example 4.10.

After applying a node-substitution, the *MGraph*-terms may be neither well-formed, nor in canonical form, hence they require some cleaning and restructuring operations.

(Propagate) $\{t \mapsto T\}N(\!|A|\!) \Longmapsto N(\!|\{t \rightsquigarrow T\}A|\!)$

(Distribute) $\{t \mapsto T\}(S_1, \ldots, S_k) \Longmapsto \{t \mapsto T\}S_1, \ldots, \{t_1 \mapsto t_2\}S_k$

(ApplySrc) $\{t \mapsto t_1, \ldots, t_n\}(i \doteq V) \Longmapsto$

$\texttt{if } id(t) \neq i \texttt{ then } i \doteq (\{t \mapsto t_1, \ldots, t_n\}V)$

$\texttt{else } id(t_1) \doteq (\{t \mapsto t_1, \ldots, t_n\}V), \ldots, id(t_n) \doteq (\{t \mapsto t_1, \ldots, t_n\}V)$

(ApplyTar) $\{t \mapsto t_1, \ldots, t_n\}(i^\frown E) \Longmapsto$

$\texttt{if } id(t) \neq i \texttt{ then } i^\frown E \texttt{ else } id(t_1)^\frown E, \ldots, id(t_n)^\frown E$

Fig. 7. Node-substitution application

$u, v : Id,\ t_1, t_2 : NeighbourSet,\ t_3, t_4 : EdgeSet,\ p, r : Port$

$(c_1)\ \bullet \doteq t_1 \to \epsilon_{AdjacencyEq}$

$(c_2)\ \bullet^\frown t_3 \to \epsilon_{Neighbour}$

$(c_3)\ (v \doteq t_1, (u^\frown t_3, (p, r), t_4), t_2) \to (v \doteq t_1, (u^\frown t_3, t_4), t_2)$

$\texttt{if } p \notin ports(v) \texttt{ or } r \notin ports(u)$

Fig. 8. Cleaning rules $\mathcal{C}$

Cleaning

Let $\mathcal{C}$ be the rewrite system defined by the cleaning rules presented in Fig. 8 which transform terms with respect the condition of well-formedness of *MGraph*-sorted terms specified in Definition 4.2 as follows: (c_1) deletes the adjacency equations for black holes; (c_2) deletes the black hole neighbours; (c_3) handles the removal of extra-edges (edges whose endpoints do not appear among the ports of the connected nodes).

The extra-edges appear as a consequence of the application of the node-substitution according to (ApplySrc) and (ApplyTar) on adjacency equations and neighbours respectively, without checking the connectivity between the new nodes. An illustration of the cleaning operation can be found in Example 4.10.

Restructuring

Let $\mathcal{R}$ be the rewrite system defined by the rules presented in Fig. 9, which transforms terms in the canonical form specified by Definition 4.3 as follows: (r_1) merges nodes having the same identifier into one node by merging their port sets; (r_2) deletes a neighbour with empty set of edges; (r_3) merges the associated sets of edges for identical neighbours; (r_4) deletes adjacency equations with empty set of neighbours; (r_5) merges adjacency equations having the same identifier in the first component into one adjacency equation by merging the sets in the second component.

$v : Id,\ n : Name,\ t_1, t_2 : PortSet,\ t_3, t_4 : NeighbourSet,\ t_5, t_6 : EdgeSet$

$(r_1)\ \langle v : n \parallel t_1\rangle, \langle v : n \parallel t_2\rangle \ \to\ \langle v : n \parallel t_1, t_2\rangle$

$(r_2)\ v^\frown \epsilon_{Edge} \ \to\ \epsilon_{Neighbour}$

$(r_3)\ (v^\frown t_5), (v^\frown t_6) \ \to\ v^\frown t_5, t_6$

$(r_4)\ v \doteq \epsilon_{Neighbour} \ \to\ \epsilon_{AdjacencyEq}$

$(r_5)\ (v \doteq t_3), (v \doteq t_4) \ \to\ v \doteq t_3, t_4$

Fig. 9. Restructuring rules $\mathcal{R}$

It is not difficult to prove (see [2]) that:

Proposition 4.8 *$\mathcal{C}$ and $\mathcal{R}$ are strongly terminating and confluent.*

We denote by $t\downarrow_{\mathcal{C}}$ and $t\downarrow_{\mathcal{R}}$ the normal forms of a term t w.r.t. the rewrite rules in $\mathcal{C}$ and $\mathcal{R}$ respectively. By Prop. 4.8 such normal forms exist and are unique.

We are now ready to define the mg-rewriting relation. Operationally, we apply extended rewrite rules, which allows us to deal only with rule application at the root position of terms.

Definition 4.9 (mg-rewriting relation) A term t of sort *MGraph* rewrites to a term t' using an mg-rewrite rule $r : t_1 \twoheadrightarrow t_2$ where $t_1, t_2 \in \mathcal{T}_{\Sigma,MGraph}(\mathcal{X})$ with $\overline{r} : \overline{t_1 \twoheadrightarrow t_2}$ and $\xi = \mathsf{GetMap}(\overline{r})$, which is denoted by $t \xrightarrow{\overline{r}} t'$, if there exists a substitution σ, a solution of the ACU-matching problem $\overline{t_1} \ll t$, such that $t' = \xi^\sigma(\sigma(\overline{t_2}))\downarrow_{\mathcal{C}}\downarrow_{\mathcal{R}}$. We call this relation *mg-rewriting* and we say that t *mg-rewrites* to t' by r.

Example 4.10 We present here a solution of mg-rewriting the term $t = \mathcal{E}(G)$ from Example 4.4 encoding the multigraph from Fig. 2, using the rewrite rule $t_1 \twoheadrightarrow t_2$ given in Example 4.6 and extended in Example 4.7 which encodes the multigraph rewrite rule (a) from Fig. 3. This mg-rewriting corresponds to the multigraph rewriting depicted in Fig. 5.

(i) one solution of the matching problem $\overline{t_1} \ll t$ is given by the substitution $\sigma = \{i \mapsto 1,\ X \mapsto A,\ x \mapsto a,\ z \mapsto b,\ j \mapsto 2,\ Y \mapsto B,\ y \mapsto e,\ W_1^p \mapsto \epsilon,\ W_2^p \mapsto c,\ W_3^p \mapsto \epsilon,\ W_4^n \mapsto (\langle 3 : A \parallel a, b, c\rangle, \langle 4 : C \parallel d\rangle),\ W_5^e \mapsto \epsilon,\ W_6^h \mapsto 3^\frown(b, a),\ W_7^a \mapsto ((3 \doteq 4^\frown(a, d)), (4 \doteq 1^\frown(d, c))\}$

(ii) $\sigma(\overline{t_2}) = (\langle 1.1 : A.1 \parallel a,\rangle, \langle 1.2 : A.2 \parallel b, c\rangle, \langle 2 : B \parallel e\rangle, \langle 3 : A \parallel a, b, c\rangle, \langle 4 : C \parallel d\rangle)(\!|\ 1.1 \doteq (2^\frown(a, e)),\ 1.2 \doteq (2^\frown(b, e)), (3^\frown(b, a)),\ (3 \doteq 4^\frown(a, d)), (4 \doteq 1^\frown(d, c))\ |\!)$ and we note the occurrence of the node identifier 1 in the adjacency equation set which is no longer a valid node identifier in this term;

(iii) $\xi^\sigma = (\langle 1 : A \parallel a, b, c\rangle) \mapsto \langle 1.1 : A.1 \parallel a\rangle, \langle 2.2 : A.2 \parallel b, c\rangle),\ (\langle 2 : B \parallel e\rangle) \mapsto \{\langle 2 : B \parallel e\rangle)$

(iv) $\xi^\sigma(\sigma(\overline{t_2})) \Longmapsto^+ (\langle 1.1 : A.1 \parallel a,\rangle, \langle 1.2 : A.2 \parallel b, c\rangle, \langle 2 : B \parallel e\rangle, \langle 3 : A \parallel a, b, c\rangle, \langle 4 : C \parallel d\rangle)(\!|\ 1.1 \doteq (2^\frown(a, e)),\ 1.2 \doteq (2^\frown(b, e)), (3^\frown(b, a)),\ (3 \doteq 4^\frown(a, d)), (4 \doteq 1.1^\frown(d, c), 1.2^\frown(d, c))\ |\!)$ by applying at the end the rule (ApplyTar) on $4 \doteq$

$1^\frown(d,c)$;

(v) $\xi^\sigma(\sigma(\overline{t_2}))\downarrow_{\mathcal{C}} = (\langle 1.1 : A.1 \parallel a,\rangle, \langle 1.2 : A.2 \parallel b,c\rangle, \langle 2 : B \parallel e\rangle, \langle 3 : A \parallel a,b,c\rangle, \langle 4 : C \parallel d\rangle)(\!|\ 1.1 \doteq (2^\frown(a,e)),\ 1.2 \doteq (2^\frown(b,e)), (3^\frown(b,a)),\ (3 \doteq 4^\frown(a,d)), (4 \doteq 1.2^\frown(d,c))\ |\!)$ using the reduction $4 \doteq 1.1^\frown(d,c), 1.2^\frown(d,c) \xrightarrow{(ExtraEdges)} 4 \doteq 1.2^\frown(d,c)$ since c does not occur in the port set of the node identified by 1.1, but occurs in the port set of the node identified by 1.2;

(vi) $\xi^\sigma(\sigma(\overline{t_2}))\downarrow_{\mathcal{C}}\downarrow_{\mathcal{R}} = \xi^\sigma(\sigma(\overline{t_2}))\downarrow_{\mathcal{C}}$ since no restructuring rule is applied.

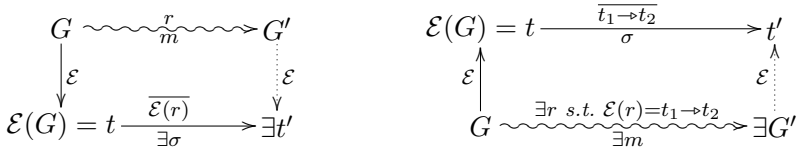
The above solution σ of the matching problem leads to the result term $\xi^\sigma(\sigma(\overline{t_2}))\downarrow_{\mathcal{C}}\downarrow_{\mathcal{R}}$ which in fact is the encoding of the multigraph G' in Fig. 5.

Proposition 4.11 *If t mg-rewrites to t' and t is a well-formed term in canonical form then t' is well-formed and in canonical form.*

Theorem 4.12 (Operational correspondence)

1) Let G, G' be two multigraphs, r a multigraph rewrite rule and m a multigraph morphism such that $G \overset{r}{\rightsquigarrow} G'$ using m. Then there exists a substitution σ and a term t' such that $\mathcal{E}(G) \overset{\overline{\mathcal{E}(r)}}{\longrightarrow\!\!\!\!\!\rightarrow} t'$ using the substitution σ and $t' = \mathcal{E}(G')$.

2) Let G be multigraph, $t = \mathcal{E}(G)$, $t_1 \twoheadrightarrow t_2$ an mg-rewrite rule, and t' such that $t \overset{\overline{t_1 \twoheadrightarrow t_2}}{\longrightarrow\!\!\!\!\!\rightarrow} t'$ with σ the solution of the matching $\overline{t_1} \ll t$ used in the rewriting. Then there exist (i) a multigraph rewrite rule r satisfying $\mathcal{E}(r) = t_1 \twoheadrightarrow t_2$, (ii) a multigraph morphism that can be constructed using σ and the structures of t and G, and (iii) a multigraph G' such that $G \overset{r}{\rightsquigarrow} G'$ using the matching morphism m and $\mathcal{E}(G') = t'$.



There is also a correspondence between all possible results of rewriting a multigraph G using a rule $G_1 \rightsquigarrow G_2$ and a morphism m, and possible results of rewriting $t = \mathcal{E}(G)$ using $t_1 \twoheadrightarrow t_2 = \mathcal{E}(G_1 \rightsquigarrow G_2)$ since all solutions of the matching $\overline{t_1} \ll t$ have as common basis the encoding of m, but different mappings for the extension variables. Hence, while the application for multigraphs of the node-substitution on unmatched partial nodes produces k results, the application of node-substitution for a term produces a term and the k solutions arise from different solutions of the ACU-matching problem with the extension variables.

5 A Multigraph Rewriting Calculus

After a short overview of the rewriting calculus (or ρ-calculus), the embedding of mg-rewriting in the rewriting calculus is presented, resulting in a multigraph term rewriting calculus.

5.1 The Rewriting Calculus

The rewriting calculus (or ρ-calculus) [10] extends first-order term rewriting and λ-calculus. From the λ-calculus, the ρ-calculus inherits its higher-order capabilities and the explicit treatment of functions and their applications. It was introduced to make all the basic ingredients of rewriting explicit objects, in particular the notions of rewrite rule (or abstraction) "$\_ \twoheadrightarrow \_$", rule application "$\_\_$", and set of results "$\_ \wr \_$". In the ρ-calculus, the usual λ-abstraction $\lambda x.t$ is replaced by a rule abstraction $T_1 \twoheadrightarrow T_2$, where T_1 and T_2 are two arbitrary terms, and the free variables of T_1 are bound in T_2.

The syntax is defined in Fig. 10 with $\mathcal{X}$ the set of variables and $\mathcal{K}$ the set of function symbols. The operator "$\_ \wr \_$" groups terms together into *structures*, and, depending on the chosen theory for this operator, it provides lists, sets or multisets to represent multiple results.

Terms	$\mathcal{T} ::= \mathcal{X} \mid \mathcal{K} \mid \mathcal{P} \twoheadrightarrow \mathcal{T} \mid \mathcal{T}\mathcal{T} \mid \mathcal{T} \wr \mathcal{T}$
Patterns	$\mathcal{P} \subseteq \mathcal{T}$

Fig. 10. The syntax of ρ-calculus

The small-step reduction semantics of the ρ-calculus is defined by the two evaluation rules in Fig. 11. If the matching problem $p \ll t_3$ has a solution σ, then the application of the rewrite rule to t_3 evaluates to $\sigma(t_2)$. The set of patterns is not a priori fixed, and the matching power of the ρ-calculus can be regulated using arbitrary theories. Therefore the semantics of the calculus depends essentially on those parameters.

(ρ)	$(p \twoheadrightarrow t_2)t_3$	$\rightarrow_\rho$	$\sigma_1(t_2) \wr \ldots \wr \sigma_n(t_2) \wr \ldots$ with $\sigma_i \in \mathcal{S}ol(p \ll t_3)$
(δ)	$(t_1 \wr t_2)t_3$	$\rightarrow_\delta$	$t_1t_3 \wr t_2t_3$

Fig. 11. The semantics of ρ-calculus

An important feature of the ρ-calculus is its capability of encoding *rewrite strategies* as shown in [11]. The basic strategies are the rewrite rules. An immediate application of the use of rewrite strategies in the ρ-calculus is the encoding of conditional rewriting [10]. The ρ-calculus has been proved confluent for linear algebraic patterns [10].

5.2 Embedding mg-Rewriting into Rewriting Calculus

Relying on the encoding of multigraph rewriting, we can now encode the mg-rewriting in the ρ-calculus as follows:

- take for $\mathcal{K}$ the operation symbols in $\mathcal{F}$ with the partial ordered set of sorts $(\mathcal{S}, <)$ and for $\mathcal{X}$ an $(\mathcal{S}, <)$-sorted family of variables;
- consider as patterns well-formed terms in canonical form in $\mathcal{T}_{\Sigma,MGraph}(\mathcal{X})$;

• consider only rewrite rules whose both sides are patterns.

We benefit in addition of the structure operator which allows grouping rules or results of applications.

As for the semantics, while (δ) dealing with the distributivity of the application over structures is taken as such from the ρ-calculus, we need a new rule for the application of a rewrite rule $t_1 \twoheadrightarrow t_2$ on a well-formed term t_3 in canonical form as follows:

$$(\rho_{mg})(\overline{t_1 \twoheadrightarrow t_2})\ t_3 \to_\rho S(\varsigma_1(\overline{t_2}))\wr\ldots\wr S(\varsigma_n(\overline{t_2}))\wr\ldots,$$
$$\texttt{if}\ \ \sigma_i \in \mathcal{S}ol(\overline{t_1} \ll t_3), \xi = \mathsf{GetMap}(\overline{t_1 \twoheadrightarrow t_2}), \varsigma_i = \sigma_i \circ \xi^{\sigma_i}$$

where $\overline{t_1 \twoheadrightarrow t_2}$ is the extended rule associated to $t_1 \twoheadrightarrow t_2$, the matching problem is solved using an ACU-matching algorithm, and S is a strategy which reduces a term to its normal form w.r.t. the cleaning and restructuring rules respectively.

We obtain this way a *rewriting calculus for labeled multigraphs with ports*, which is an instance of ρ-calculus, and we call it the ρ_{mg}-calculus.

6 Conclusion

An implementation for the multigraphs with ports is currently developed in TOM [1] [6] using pointers for the termgraph implementation [5] which handles cyclic termgraphs as well. We use a more efficient encoding for programming by representing a multigraph by its node set and each edge by a pointer to the target port associated to the source port. The use of pointers and TOM's maximal sharing is very important in order to cope with computer representation of large terms.

The generality of the notion of multigraph with ports allows expressing different types of multigraphs, from simple graphs to multigraphs with ports with states. For further applications, an interesting research direction is to enhance multigraphs with ports with a hierarchical node structure describing nested nodes, and to relate them to Milner's bigraphs [17].

References

[1] Andrei, O., L. Ibanescu and H. Kirchner, *Non-intrusive Formal Methods and Strategic Rewriting for a Chemical Application*, in: K. Futatsugi, J.-P. Jouannaud and J. Meseguer, editors, *Algebra, Meaning, and Computation, Essays Dedicated to Joseph A. Goguen on the Occasion of His 65th Birthday*, LNCS **4060** (2006), pp. 194–215.

[2] Andrei, O. and H. Kirchner, *A Rewriting Calculus for Labeled Multigraphs with Ports*, Long version available at http://hal.inria.fr/inria-00139363/en/.

[3] Andrei, O. and H. Kirchner, *Rewrite Rules and Strategies for Molecular Graphs*, Submitted, May 2007.

[4] Baader, F. and T. Nipkow, "Term Rewriting and All That", Cambridge University Press, 1998.

[5] Balland, E. and P. Brauner, *Term-graph rewriting in Tom using relative positions*, in: *Pre-proceedings of 4th International Workshop on Computing with Terms and Graphs - TERMGRAPH*, 2007.

[6] Balland, E., P. Brauner, R. Kopetz, P.-E. Moreau and A. Reilles, *Tom: Piggybacking Rewriting on Java*, in: *To appear in Proceedings of RTA*, 2007.

[1] Web page http://tom.loria.fr.

[7] Blinov, M. L., J. Yang, J. R. Faeder and W. S. Hlavacek, *Graph theory for rule-based modeling of biochemical networks*, in: C. Priami, A. Ingólfsdóttir, B. Mishra, and H. R. Nielson, editors, *T. Comp. Sys. Biology VII*, LNCS **4230** (2006), pp. 89–106.

[8] Bournez, O., L. Ibanescu and H. Kirchner, *From Chemical Rules to Term Rewriting*, ENTCS **147** (2006), pp. 113–134.

[9] Brandes, U., M. Eiglsperger, I. Herman, M. Himsolt and M. S. Marshall, *GraphML Progress Report: Structural Layer Proposal*, in: *Graph Drawing*, LNCS **2265** (2002), pp. 501–512.

[10] Cirstea, H. and C. Kirchner, *The rewriting calculus - Part I and II*, Logic Journal of the IGPL **9** (2001), pp. 427—498.

[11] Cirstea, H., C. Kirchner, L. Liquori and B. Wack, *Rewrite strategies in the rewriting calculus*, ENTCS **86** (2003), pp. 593–624.

[12] Corradini, A., U. Montanari, F. Rossi, H. Ehrig, R. Heckel and M. Löwe, *Algebraic Approaches to Graph Transformation - Part I: Basic Concepts and Double Pushout Approach*, in: G. Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*, World Scientific, 1997 pp. 163–246.

[13] Danos, V. and C. Laneve, *Formal molecular biology*, TCS **325** (2004), pp. 69–110.

[14] Goguen, J. A. and J. Meseguer, *Order-Sorted Algebra I: Equational Deduction for Multiple Inheritance, Overloading, Exceptions and Partial Operations*, TCS **105** (1992), pp. 217–273.

[15] Kirchner, C. and H. Kirchner, *Rewriting, Solving, Proving*, Available at http://www.loria.fr/~ckirchne/=rsp/rsp.pdf.

[16] Kirchner, H. and P.-E. Moreau, *Promoting rewriting to a programming language: a compiler for non-deterministic rewrite programs in associative-commutative theories*, J. Funct. Program. **11** (2001), pp. 207–251.

[17] Milner, R., *Bigraphical Reactive Systems*, in: K. G. Larsen and M. Nielsen, editors, *CONCUR'01*, LNCS **2154** (2001), pp. 16–35.

[18] Schürr, A., *Programmed Graph Replacement Systems*, in: G. Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*, World Scientific, 1997 pp. 479–546.